Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nick Jokić

Realnočasovna vizualizacija distribucijskih električnih omrežij v spletnem okolju

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel
Somentor: izr. prof. dr. Aleš Švigelj

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Realnočasovna vizualizacija distribucijskih električnih omrežij v spletnem okolju

Tematika naloge:

Porazdeljena proizvodnja električne energije, kjer so njeni uporabniki lahko tudi njeni proizvajalci, postaja vse bolj aktualna. Zaradi dvosmernega pretoka moči pa se pojavljajo vprašanja o stabilnosti napetosti omrežja, zaradi česar je realnočasovni prikaz stanja omrežja izjemnega pomena, saj omogoča takojšnje odkrivanje nestabilnosti v omrežju. V okviru diplome je vaša naloga, da razvijete spletno aplikacijo, ki bo omogočala prikaz podatkovnih tokov meritev v realnem času in z zadosti veliko frekvenco posodabljanja prikaza podatkovnih tokov meritev. Razvita aplikacija mora podpirati izbor in prikaz različnih podomrežij, grafični prikaz izbranega podomrežja na zemljevidu, vizualizacijo različnih realnočasovnih podatkovnih tokov in vizualizacijo preteklih meritev. Pri izdelavi spletne aplikacije izberite najprimernejše tehnologije na strani strežnika in odjemalca. Pri vizualizaciji tokov podatkov se osredotočite predvsem na učinkovito realizacijo realnočasovnega prikaza tokov podatkov, pri katerih podatki prihajajo z visoko frekvenco. Preučite različne možnosti za izris, ki jih imate na voljo, in za posamezne vizualizacije izberite najprimernejše možnosti.

Zahvaljujem se mentorju doc. dr. Alešu Smrdelu in somentorju izr. prof. dr Alešu Šviglju za strokovno vodenje in pomoč pri izdelavi diplomskega dela. Zahvaljujem se tudi Urbanu Kuharju in dr. Kemalu Aliču za sodelovanje, pomoč in produktivno delovno okolje med delom na projektu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
DSSE	distribution system state estimator	ocenjevalnik stanja distribucijskih sistemov
GPS	global positioning system	globalni sistem pozicioniranja
PMU	phasor measurement unit	enota za merjenje fazorjev
DFT	discrete Fourier transform	diskretna Fourierova transformacija
HTML	hyper text markup language	označevalni jezik
XML	extensible markup language	razširljivi označevalni jezik
CSS	cascading style sheets	kaskadne stilske predloge
SVG	scalable vector graphics	umerljiva vektorska grafika
DOM	document object model	objektni model dokumenta
AJAX	asynchronous JavaScript and XML	asinhroni JavaScript in XML
API	application programming interface	aplikacijski programski vmesnik
IoT	internet of things	internet stvari
JSON	JavaScript object notation	notacija objekta JavaScript
MQTT	message queuing telemetry transport	protokol za prenos telemetrijskih sporočil
NPM	Node package manager	Node upravitelj paketov

Povzetek

Naslov: Realnočasovna vizualizacija distribucijskih električnih omrežij v spletnem okolju

Avtor: Nick Jokić

Živimo v svetu, kjer imajo podatki velik vpliv na vsakdanje življenje. Vizualizacija podatkov je pomemben mehanizem, saj ljudje vizualne interpretacije hitreje in lažje razumemo kot tekstovne. Danes je še posebej koristna realnočasovna vizualizacija, saj omogoča upodobitev podatkovnih tokov iz različnih virov tisti trenutek, ko so na voljo, kar omogoča sprotni pregled in takojšnje reagiranje na anomalije.

S kontinuiranim razvojem distribucijskih in pametnih električnih omrežij se veča potreba po opazovanju stanj omrežij v realnem času. Realnočasovno opazovanje distribucijskih sistemov je ključno za učinkovito upravljanje in zagotavljanje nemotenega delovanja omrežij v prihodnosti. V diplomskem delu predstavimo cilje evropskega projekta SUNSEED ter teoretično ozadje distribucijskih električnih omrežij. Osredotočamo se v razvoj spletne aplikacije, ki nudi učinkovito realnočasovno vizualizacijo podatkovnih tokov različnih merilnih naprav znotraj distribucijskih električnih omrežij. Takšna aplikacija omogoča upravljavcem distribucijskih sistemov lažje opazovanje posameznih sistemov. V diplomskem delu obravnavamo problem visokofrekvenčnih realnočasovnih vizualizacij v spletnem okolju in predlagamo različne optimizacijske pristope. Podrobneje predstavimo tudi implementacijo končne aplikacije, ki združuje veliko različnih funkcionalnosti in gradnikov.

Ključne besede: spletni vmesnik, distribucijska električna omrežja, realnočasovna

vizualizacija, enota za merjenje fazorjev, ocenjevalnik stanja, SUNSEED.

Abstract

Title: Real-time web-based visualization of distribution electricity grids

Author: Nick Jokić

We live in a world where data has a major impact on everyday life. Visualization of data is an important mechanism since people much quicker and more easily understand visual interpretations than textual interpretations. An enormous value nowadays has real-time visualization, as it enables the representation of data flows from different sources as soon as they are available, allowing for prompt inspection and immediate response to anomalies.

With the continuous development of distribution and smart electricity grids, the need for real-time observation of grid states is growing. Real-time observation of distribution systems is essential for effective management and ensuring smooth operation of networks in the future. In the diploma thesis we present the objectives of the European project SUNSEED and the theoretical background of distribution electricity grids. We focus on the development of a web application that offers efficient real-time visualization of data streams of various measuring devices within distribution electricity grids. Such an application enables distribution system operators to easily monitor individual systems. In the diploma thesis we deal with the problem of high-frequency real-time visualizations within a web-based application and propose different optimization techniques. We also present in greater detail the implementation of the final application, which contains many different components and functionalities.

Keywords: web interface, distribution electricity grids, real-time visualization,

phasor measurement unit, state estimator, SUNSEED.

Poglavje 1

Uvod

Porazdeljena proizvodnja električne energije je v zadnjih letih deležna precejšnega zanimanja. S pojavom dvosmernega pretoka v sodobnih distribucijskih omrežjih se znatno povečuje tudi soproizvodnja električne energije iz obnovljivih virov. V takšnih distribucijskih sistemih je ključen korak za učinkovitejše operativne zmogljivosti uvedba zmožnosti opazovanja distribucijskih sistemov.

Izhodišče diplomskega dela je potreba po vizualizaciji distribucijskih električnih omrežij v spletnem okolju. Ta omogoča upravljavcem distribucijskih sistemov lažje opazovanje stanj posameznih sistemov v realnem času, kar je ključnega pomena za odkrivanje anomalij v omrežju in izboljšanje kakovosti električne energije.

1.1 Motivacija

Glavna motivacija je bila razvoj spletne aplikacije, ki bi omogočala poenostavljen realnočasovni prikaz stanj distribucijskih električnih omrežij. Kljub hitremu in nenehnemu razvoju distribucijskih in pametnih električnih omrežij se realnočasovni vizualizaciji v spletnih okoljih še vedno ne namenja dovolj pozornosti. Primarni vzrok je pomanjkanje standardiziranega načina implementacije različnih funkcionalnosti v celovito in performančno učinkovito spletno aplikacijo.

1.2 Cilji

Osrednji cilj diplomskega dela je prikazati način izgradnje celovite spletne aplikacije, ki upravljavcem distribucijskih sistemov omogoča opazovanje trenutnega in preteklega stanja omrežja. Rezultat diplomskega dela je delujoča spletna aplikacija, ki združuje vse funkcionalnosti vizualizacije distribucijskih električnih omrežij v celostno rešitev. Ugotovitve te naloge bodo še posebej koristile vsem, ki raziskujejo področje pametnih električnih omrežij in stremijo k implementaciji učinkovite realnočasovne vizualizacije podatkov v spletnih okoljih, kar upravljavcem distribucijskih sistemov omogoča celovit vpogled v stanje omrežja.

1.3 Struktura diplomskega dela

V 2. poglavju so opisani cilji projekta SUNSEED [20] ter teoretično ozadje distribucijskih električnih omrežij. Predstavljene so tudi uporabljene merilne naprave in matematični modeli, ki v aplikaciji predstavljajo osrednji vir kasneje vizualiziranih meritev.

V 3. poglavju sledi pregled spletnih tehnologij, ki so bile uporabljene za učinkovito implementacijo realnočasovne aplikacije.

V 4. poglavju sta predstavljeni osrednji tehnologiji, ki se najpogosteje uporabljata za namen vizualizacije v spletnih okoljih. Predstavljen je tudi način implementacije realnočasovne vizualizacije z uporabo knjižnic JavaScript. Za ponazoritev so v poglavju predstavljeni različni primeri realnočasovnih vizualizacij. Poglavje je osredotočeno na različne optimizacijske postopke, ki se lahko učinkovito uporabijo v praksi.

V 5. poglavju je predstavljena in podrobneje opisana končna aplikacija, ki vsebuje številne različne gradnike. Predstavljen je potek komunikacije med strežnikom in odjemalcem, grafični vmesnik ter podroben opis implementacije posameznih gradnikov.

V 6. poglavju so povzete sklepne ugotovitve naloge, navedene pa so tudi ideje za možne nadgradnje in nadaljnje delo v prihodnosti.

Poglavje 2

Ozadje projekta in opis problemske domene

SUNSEED [20] je triletni evropski razvojni projekt, ki se je pričel februarja 2014. Sestavlja ga mednarodni konzorcij devetih partnerjev iz šestih držav. Na tem projektu sem sodeloval od avgusta 2016 do zaključka projekta v juniju 2017.

Razvojni projekt SUNSEED se ukvarja z raziskovanjem trajnostnih in robustnih omrežij za pametno distribucijo električne energije. Predlaga učinkovito povezavo že obstoječih komunikacijskih omrežij v konvergirano komunikacijsko infrastrukturo za bodoča pametna energetska omrežja. Namen projekta je postavitev enotne komunikacijske infrastrukture in opredelitev poslovnega modela v bodočih pametnih električnih omrežjih (angl. smart grids). Eden izmed glavnih ciljev projekta je vzpostavitev platforme za napredno upravljanje distribucijskega omrežja, ki omogoča spremljanje napetostnih profilov in pretokov moči v realnem času s pomočjo naprednih merilnih sistemov (angl. wide area measurement systems), ki izvajajo meritve in generirajo vhodne podatke za ocenjevalnik stanja (DSSE). Projekt obsega tudi postavitev testnega poligona na različnih lokacijah (npr. Kromberk in Kneža), kjer ima vsaka lokacija drugačne značilnosti. Med glavnimi cilji projekta sta tudi razvoj in implementacija inteligentnih analitičnih in vizualnih orodij za upravljanje pametnih omrežij v realnem času [26, 34, 31, 20].

2.1 Distribucijska električna omrežja

Distribucijska električna omrežja prenašajo električno energijo iz visokonapetostnega prenosnega omrežja do različnih končnih uporabnikov. V preteklosti so bila oblikovana za enosmerni pretok moči. Del distribucijskega omrežja je bil priključen na prenosno omrežje, moč pa se je prenesla v nizkonapetostna omrežja h končnim uporabnikom. Model centralizirane proizvodnje, ki pogostokrat zahteva prenos električne energije preko velikih razdalj, je standardni način proizvodnje električne energije po vsem svetu. Razlogi, kot so globalne podnebne spremembe in prihod električnih avtomobilov, zaradi katerih se pričakuje znatno povečana poraba električne energije, so ustvarili potrebo po zmanjšanju odvisnosti od proizvodnje električne energije iz klasičnih virov, kot sta premog in jedrska energija. Posledično postajajo tehnologije, povezane s porazdeljenimi obnovljivimi viri in porazdeljeno proizvodnjo, z vsakim dnem pomembnejše [27].

Porazdeljena proizvodnja električne energije (angl. distributed generation) je decentraliziran pristop, ki omogoča proizvodnjo električne energije v bližini končnih uporabnikov s pomočjo manjših proizvodnih virov. V zadnjem desetletju se posveča porazdeljeni proizvodnji vse večji interes. Ljudje v svojih domovih in podjetjih nameščajo razne tehnologije obnovljivih virov (npr. solarni sistemi, kogeneracija, vetrne turbine), s katerimi proizvajajo del svoje lastne električne energije. Dvosmerni pretok v sodobnih distribucijskih omrežjih omogoča potovanje odvečne proizvedene električne energije s strani uporabnikov, ki so proizvajalci in potrošniki hkrati (angl. prosumers), nazaj v omrežje, kar zagotavlja energijo ostalim lokalnim uporabnikom.

Porazdeljena proizvodnja prispeva k izboljšanju kakovosti električne energije v obliki stabilnejših napetosti, manj pogostih napetostnih kršitev (angl. voltage violations) in primernega razmerja med jalovo in delovno energijo. Napetostne kršitve poimenujemo pojav, ko je napetost izven dovoljenih meja (230V $\pm$10%). Kljub temu pa se pojavljajo vprašanja o morebitnih nestabilnostih omrežij, ki se lahko pojavijo z dvosmernim pretokom moči. S širitvijo porazdeljene proizvodnje pa se porajajo tudi razni ekonomski izzivi, kot na primer stroškovno učinkovita uvedba naprednih merilnih sistemov in zagotovitev stroškovno učinkovitega omrežja. Ključen

korak k učinkovitejšim operativnim zmogljivostim je uvedba zmožnosti opazovanja distribucijskega sistema. V nadaljevanju so predstavljene napredne merilne naprave, ki omogočajo upravljavcem distribucijskih sistemov nadzor in vpogled nad pretoki energije na omrežju s srednjo/nizko napetostjo [4, 32, 31].

2.2 Merilne naprave

Napredne merilne naprave, nameščene v distribucijskem omrežju, so ključnega pomena, saj omogočajo opazovanje in nadzor nad stanjem v omrežju. Nekaj pomembnih koristi poznavanja trenutnega stanja omrežja so [31]:

- motnje na nižji napetosti se lahko zaznajo in odpravijo še preden pride do vpliva na druge dele sistema;
- upravljavcem distribucijskih sistemov omogoča s pomočjo identificiranih primanjkljajev v omrežju izdelavo natančnih modelov, primernih za načrtovanje in analize;
- omogoča razvoj realnočasovnega nadzora omrežij.

V nadaljevanju so predstavljene merilne naprave in matematični modeli, ki v sklopu projekta SUNSEED predstavljajo ključne vire meritvenih podatkov, kasneje uporabljenih v različnih vizualizacijah.

2.2.1 PMU

Enote za merjenje fazorjev (PMUs) so namenske naprave s skupno časovno referenco, ki jih zagotavlja natančna ura (GPS). Omogočajo časovno sinhronizirano merjenje fazorjev na različnih lokacijah. Fazor je kompleksno število, ki predstavlja magnitudo, frekvenco ter fazni kot sinusnih valov. Visoka natančnost in hitrost vzorčenja meritev napetostnih ali tokovnih fazorjev v vseh treh fazah na različnih zbiralkah (iz več enot PMU) omogočata celovit pregled nad stanjem celotnega omrežja. Meritve fazorjev, ki so istočasno pridobljene, imenujemo sinhrofazorji (angl. synchrophasors). Algoritmi za analizo signalov (npr. DFT) omogočajo

izračun količin, kot so magnituda in fazni koti električne napetosti iz vzorčenega signala. Za potrebe projekta SUNSEED je bila za ta namen razvita naprava μPMU (slika 2.1) [31].

Slika 2.1: Slika naprave μPMU, ki je bila razvita v sklopu projekta SUNSEED [31].

Naprave μPMU so povezane preko komunikacijskega omrežja LTE in omogočajo komunikacijo z zalednimi sistemi s pomočjo komunikacijskega protokola MQTT. V realnem času se, poleg statusnih informacij, posredujejo različne meritve (slika 2.2), kot so trenutna efektivna vrednost magnitud, frekvence in fazni koti. Te meritve so kasneje uporabljene pri vizualizaciji.

```
{ "node_id": "167002045410006104c5a000a00000f5",
  "week_id": 1949,
  "sec_id": 88660,
  "report_n": 0,
  "v1": 59.03785627,
  "v2": 59.14799264,
  "v3": 59.13165254,
  "psp_v": 0.03307204,
  "th1": 1.41131878,
  "th2": -2.77725568,
  "th3": -0.68313475,
  "psp_th": -1.5218976,
  "f1": 50.00178296,
  "f2": 50.00222598,
  "f3": 50.00184605,
  "f4": 50.00195166,
  "status1": 0,
  "status2": 0,
  "status3": 0,
  "status4": 0,
  "GPSstatus": 0 }
```


Slika 2.2: Podatki v formatu JSON, ki so poslani s strani naprave μPMU preko komunikacijskega protokola MQTT.

2.2.2 DSSE

Ocenjevalnik stanja distribucijskih sistemov (DSSE) je program, ki omogoča izračun stanja celotnega omrežja iz omejenega števila meritev in topologije omrežja, hkrati pa lahko prepozna napake v meritvah v omrežju. Je ključnega pomena za zanesljivo in natančno poznavanje stanja omrežja. Prednost DSSE je, da lahko upošteva vse vrste razpoložljivih meritev in s tem zmanjša naložbene stroške, ki bi nastali z zahtevano merilno infrastrukturo. Natančnost DSSE je odvisna od mnogo faktorjev, kot so gostota in lokacije enot PMU, točnost in interval poročanja merilne infrastrukture v omrežju. Vir meritev za DSSE so v samem projektu predstavljale naprave μPMU, pametni števci (angl. smart meters) in naprave za merjenje in nadzor moči (angl. power measurements and control devices). DSSE za vsako zbiralko (angl. bus) izračuna napetost in fazni kot, ki se uporabljata v nadaljnjih izračunih za količine, kot so delovna moč (P), jalova moč (Q), navidezna moč (S) in faktor moči (φ). Omenjeni izračuni potekajo za vsako vejo v omrežju, kar upravljavcem distribucijskih sistemov omogoča sklepanje na morebitne preobremenitve na posameznih povezavah v omrežju. Spremljanje napetosti na vsaki zbiralki omogoča izdelavo napetostnega profila (angl. voltage profile), ki ima pomembno vlogo tudi pri odkrivanju napetostnih kršitev v omrežju [31].

2.3 Zahtevane funkcionalnosti aplikacije

V sklopu projekta SUNSEED se je pojavila potreba po razvoju spletne aplikacije, ki ima vlogo nadzorne plošče oz. vizualizacijskega orodja, ki nudi takojšen vpogled v trenutno in preteklo stanje distribucijskega električnega omrežja. Definirali smo naslednje zahteve aplikacije:

- aplikacija naj podpira več različnih podomrežij ter omogoča njihov izbor;
- aplikacija naj prikaže (geografsko natančen) zemljevid trenutno izbranega omrežja z vsemi vozlišči (zbiralkami) in povezavami;
- zemljevid naj je interaktiven in naj omogoča izbor različnih vozlišč in njihovih

podatkovnih tokov (meritev PMU in izračunov DSSE) ter prikaz trenutnega pretoka moči na povezavah;

- aplikacija naj omogoča visokofrekvenčno realnočasovno vizualizacijo različnih podatkovnih tokov (meritev PMU);
- aplikacija naj omogoča realnočasovni prikaz podatkov v tekstovni obliki (izračunov DSSE);
- aplikacija naj omogoča realnočasovno vizualizacijo napetostnega profila, ki potrebne razdalje med vozlišči izračuna iz zemljevida;
- aplikacija naj omogoča vizualizacijo velikega števila preteklih meritev, ki se hranijo v podatkovni bazi MongoDB.

V naslednjem poglavju so podrobneje opisane uporabljene tehnologije, ki so bile prepoznane kot primerne za učinkovito doseganje definiranih zahtev.

Poglavje 3

Pregled tehnologij

Končna spletna aplikacija vsebuje strežniški in odjemalni del, ki uporabljata različne tehnologije in veliko različnih knjižnic, ki so podrobneje opisane v nadaljevanju.

3.1 Tehnologije na strani odjemalca

Na strani odjemalca so uporabljene naslednje tehnologije:

- HTML;
- CSS;
- JavaScript;
- jQuery;
- Bootstrap;
- Rickshaw.js, Morris.js, Dygraphs in Function Plot;
- Google Maps API.

3.1.1 HTML

HTML (angl. Hypertext Markup Language) je označevalni jezik, ki se kot jedrna (angl. core) tehnologija uporablja v spletnih aplikacijah in straneh. Osrednji

sestavni del sintakse HTML so oznake (angl. tags), ki jih pišemo znotraj znaka za manjše in večje (npr. <a>). Lahko so bodisi samostojne (npr. <img/>) bodisi sestavljene iz začetne in končne oznake (npr.). Oznake lahko vsebujejo vsebino, ki lahko obsega ostale oznake (gnezdenje) in besedilo, hkrati pa imajo lahko opredeljene različne atribute (npr. class, id, style itd.).

Dokument HTML se običajno začne z deklaracijo tipa dokumenta (angl. DOCTYPE), ki brskalniku pove, za kateri tip dokumenta gre. Sledi mu oznaka <html>, ki predstavlja korensko oznako dokumenta HTML in je starš oz. vsebnik vsem ostalim elementom HTML (razen deklaraciji tipa dokumenta). Pomembni oznaki sta še <head> in <body>. Oznaka <head> je obvezen del dokumenta, katere vsebina ni vidna na sami strani. Vsebuje osnovne informacije o dokumentu (npr. naslov strani, razni metapodatki itd.), vstavi pa se lahko tudi različno kodo, kot na primer kodo CSS oziroma povezave na datoteke CSS. Oznaka <body>, ki je prav tako obvezna, deklarira telo dokumenta, ki je vsebinski del dokumenta. Vsebina je prikazana v obliki besedil, nadpovezav, seznamov, slik itd.

HTML5 je trenutno zadnja večja različica standarda HTML, ki je bila izdana oktobra 2014 s strani organizacije W3C (angl. World Wide Web Consortium). HTML5 je prinesel veliko izboljšav in novosti na področjih semantike, multimedije, 2D in 3D grafike, performančne učinkovitosti in še na mnogo drugih področjih [8].

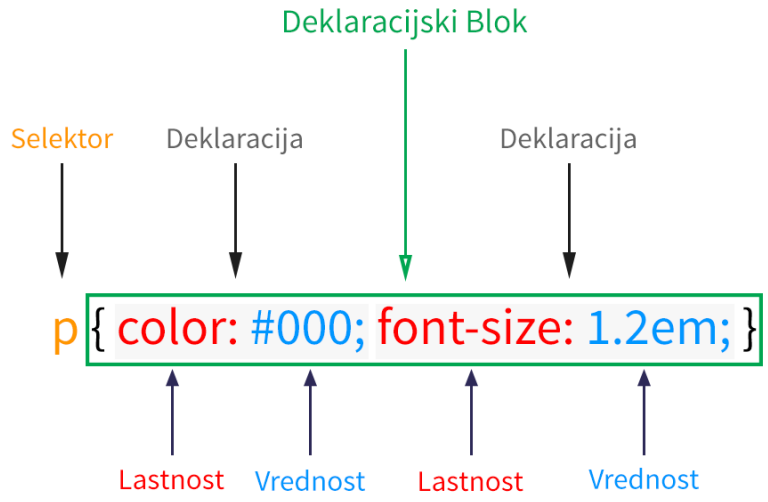
3.1.2 CSS

CSS oz. kaskadne stilske predloge skrbijo za stiliranje dokumenta HTML. Omogočajo pisanje pravil, ki so sestavljena iz selektorja, deklaracijskega bloka in deklaracije, ki vsebuje lastnost ter vrednost. Selektor pove brskalniku, kateri element na strani stiliramo. Razlikujemo med več različnimi selektorji, ki so si sintaktično različni:

- selektor tipa (npr. 'p');
- selektor razreda (npr. '.imeRazreda');
- selektor identifikacije (npr. '#IDime');
- univerzalni selektor (npr. '*');

- selektor atributa (npr. 'input[type="text"]');
- psevdorazred (npr. 'a:hover');
- psevdoelement (npr. '::after').

Deklaracijski blok stoji takoj za definicijo selektorja in je sestavljen iz začetnega zavitega oklepaja, deklaracije in končnega zavitega oklepaja. V deklaracijskem bloku je možno navesti več različnih deklaracij, ki se jih medsebojno loči s podpičjem. V deklaraciji nastopa lastnost, ki jo izberemo iz množice končno mnogo definiranih lastnosti in ponazarjajo lastnost elementa HTML, ki jo spreminjamo. Za lastnostjo dodamo vrednost, ki je od lastnosti ločena z dvopičjem in ponazarja specifično nastavitev za definirano lastnost [29]. Slika 3.1 prikazuje sintakso določanja lastnosti za elemente HTML, izbrane s poljubnim selektorjem.



Slika 3.1: Osnovna struktura pravila CSS.

S prihodom CSS3, ki je trenutno najnovejša različica stilskega jezika CSS, so prišle nove koristne funkcionalnosti, kot so animacije, prehodi, gradienti, 3D transformacije in druge.

3.1.3 JavaScript in jQuery

JavaScript [9] je objektno orientiran skriptni programski jezik, ki omogoča manipulacijo strukture dokumenta HTML ter s tem omogoča dodajanje dinamične komponente spletnim stranem oz. aplikacijam. Večina naprav in sodobnih brskalnikov danes podpira JavaScript. V osnovi je bil JavaScript implementiran za kodiranje na strani odjemalca, sedaj pa ga je možno uporabljati tudi kot strežniško tehnologijo (npr. Node.js), uporablja pa se tudi v nekaterih podatkovnih bazah (npr. MongoDB) [28].

JQuery [10] je odprtokodna knjižnica JavaScript, ki je bila ustvarjena z namenom poenostavitve pisanja kode JavaScript po principu "napiši manj - naredi več". Omogoča poenostavljeno manipulacijo DOM in selekcijo elementov HTML, kreiranje animacij, rokovanje dogodkov, izvedbo klicev AJAX itd. JQuery sodi med najbolj uporabljene knjižnice JavaScript na spletu in je deležen velike podpore skupnosti. Knjižnica med drugim omogoča tudi kreiranje lastnih vtičnikov (angl. plug-ins) [28].

3.1.4 Bootstrap

Bootstrap [1] je eno izmed zelo priljubljenih odprtokodnih ogrodij HTML, CSS in JavaScript. Uporablja se pri razvoju odzivnih oziroma *mobile-first* spletnih vmesnikov aplikacij. Osrednje funkcionalnosti ogrodja Bootstrap so 12-stolpični mrežni sistem, osnovne stilske predloge CSS, vnaprej oblikovane komponente (spustni meniji, plošče, navigacijski stolpci itd.) in komponente JavaScript (namigi, opozorila, modalna okna itd.). Glavne prednosti ogrodja Bootstrap so:

- hitro prototipiranje novih načrtov;
- dobra podprtost sodobnih brskalnikov;
- omejitev ponavljanja kode;
- spodbujanje konsistentnosti med projekti;
- velika podpora skupnosti.

3.1.5 Rickshaw.js, Morris.js, Dygraphs in Function Plot

Rickshaw.js [18] je odprtokodno vizualizacijsko orodje, ki temelji na knjižnici D3.js ter omogoča ustvarjanje različnih interaktivnih grafikonov. Pri izrisovanju grafikonov uporablja tehnologijo SVG, kar mu kasneje omogoča stiliranje s CSS. Podpira vrsto različnih tipov grafikonov (npr. črtni, stolpični, površinski), izris interaktivne legende, drsnike in anotacije.

Morris.js [12] je vizualizacijsko orodje, ki temelji na knjižnicah jQuery in Raphael.js [17] ter omogoča ustvarjanje različnih odzivnih in interaktivnih grafikonov. Pri risanju grafikonov prav tako uporablja tehnologijo SVG.

Dygraphs [3] je odprtokodno vizualizacijsko orodje, ki ponuja ustvarjanje interaktivnih vizualizacij časovnih vrst. Posebej primerno je za vizualizacijo velikega nabora podatkov. Omogoča povečavo in premik po množici podatkov, označitev posamezne vrednosti z miško in določitev časovnega obdobja. Pri izrisovanju grafikonov uporablja tehnologijo Canvas.

Function Plot [6] je odprtokodna vizualizacijska knjižnica zgrajena na vrhu D3.js in se uporablja za prikazovanje matematičnih funkcij. Podpira interaktivno povečavo, premikanje po neskončnemu grafikonu in vizualizacijo mnogih funkcij znotraj enega grafikona. Prav tako poleg izrisa parametričnih in implicitnih oblik enačb omogoča tudi učinkovit prikaz vektorjev v 2D prostoru.

3.1.6 Google Maps API

Google Maps API [7] je aplikacijski programski vmesnik, katerega avtor je podjetje Google. Namenjen je vgradnji zemljevidov na spletne strani in tako omogoča uporabo interaktivnih zemljevidov na različnih sistemih. S pomočjo različnih storitev (angl. services) omogoča načrtovanje poti med različnimi lokacijami, dodajanje lastnih markerjev, izračun razdalj, pridobivanje podatkov o višinah za lokacije na površini zemlje in drugo.

3.2 Tehnologije na strani strežnika

V aplikaciji so uporabljene naslednje strežniške tehnologije:

- Node.js in Express.js;
- Socket.io;
- MQTT.js;
- MongoDB.

3.2.1 Node.js in Express.js

Node.js [15] je asinhrona odprtokodna programska platforma, ki v jedru vsebuje interpreter JavaScript in se uporablja za ustvarjanje enonitnih strežniških aplikacij. Uporablja asinhron dogodkovno orientiran model, ki je zelo uporaben pri sočasnem ravnanju z zahtevami. Večina kode teče na principu povratnih klicev (angl. callbacks), ki preprečijo zaklepanje aplikacije v primeru več sočasno nedokončanih dogodkov. Sestavlja ga pogon JavaScript Google V8, ki je napisan v jeziku C. Node.js vsebuje tudi upravitelja paketov NPM, ki skrbi za enostavno upravljanje z različnimi knjižnicami. Node.js je zaradi svojih lastnosti še posebej priljubljen v realnočasovnih aplikacijah [16, 24].

Express.js [5] je minimalistično odprtokodno aplikacijsko ogrodje, ki razširja Node.js in je namenjen poenostavljenemu ter hitrejšemu razvoju strežniških aplikacij in APIjev. Express.js je dejansko standardno strežniško ogrodje za Node.js.

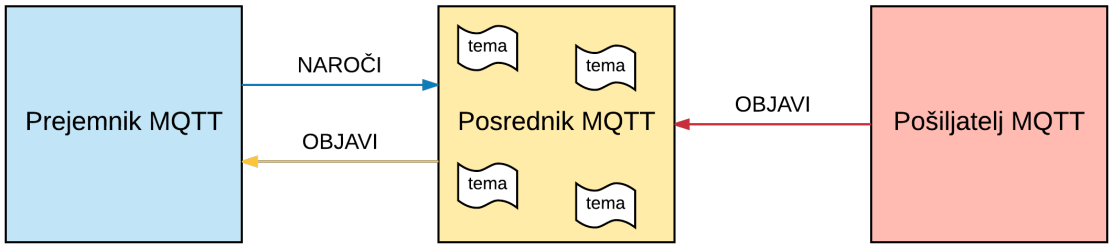
3.2.2 Socket.io

Socket.io [19] je knjižnica JavaScript, ki temelji na protokolu WebSocket in omogoča dvosmerno asinhrono dogodkovno-orientirano komunikacijo med strežnikom in odjemalcem. Zagotavlja sloj abstrakcije nad WebSocketom tako za strežniški Node.js kot tudi za JavaScript na strani odjemalca. Poleg ovoja za WebSocket omogoča tudi shranjevanje podatkov povezanih z odjemalci ter oddajanje v več vtičnic (angl.

sockets). Pogostokrat se uporablja za klepetalnice, realnočasovne analitike, video konference in spletne večigralske igre [25].

3.2.3 MQTT.js

MQTT.js [14] je knjižnica JavaScript, ki omogoča komunikacijo aplikacije Node.js z oddaljenimi napravami preko komunikacijskega protokola MQTT. Komunikacijski protokol MQTT deluje po principu objavi-naroči (angl. publish-subscribe). Ta način komunikacije omogoča enostavno ločitev pošiljatelja od prejemnika. Pošiljatelj objavlja sporočila v ustrezne teme, na katere se prejemnik naroči. Med pošiljateljem in posrednikom obstaja tudi tretja komponenta, ki jo imenujemo posrednik (angl. broker). Posrednik je primarno odgovoren za sprejemanje vseh sporočil, filtriranje in pošiljanje sporočil vsem naročnikom. Posrednik opravlja tudi avtentikacijo in avtorizacijo odjemalcev [13]. Potek komunikacije je prikazan na sliki 3.2.



Slika 3.2: Potek komunikacije s protokolom MQTT, ki poteka med pošiljateljem, posrednikom in prejemnikom.

3.2.4 MongoDB

MongoDB [22] je odprtokodna nerelacijska (NoSQL) podatkovna baza, ki se pogostokrat uporablja pri hitrem iterativnem razvoju aplikacij, kjer sta pomembni skalabilnost in učinkovitost. Omogoča fleksibilno hranjenje podatkov, saj ne zahteva vnaprejšnjih definicij shem. MongoDB namesto tabel in vrstic uporablja arhitekturo zbirk (angl. collections) in dokumentov. Zbirke vsebujejo množico dokumentov, ki imajo dinamične sheme ter vsebujejo podatke v obliki ključ-vrednost

(angl. key-value) parov. Omogoča učinkovite načine dostopanja in analizo podatkov s pomočjo indeksiranja, ad hoc poizvedb ter združevanja v realnem času [11].

Poglavje 4

Vizualizacija podatkov z JavaScript

V tem poglavju so predstavljene tehnike in tehnologije povezane z vizualizacijo podatkov na spletnih straneh. Za ponazoritev so v poglavju predstavljeni različni primeri realnočasovnih vizualizacij in različni optimizacijski postopki, ki pripomorejo k učinkovitejši vizualizaciji visokofrekvenčnih podatkovnih tokov.

4.1 SVG in Canvas

SVG in Canvas sta osrednji tehnologiji, ki se uporabljata pri neposrednem izrisovanju 2D vsebin v spletnih brskalnikih brez uporabe vtičnikov. Omogočata implementacijo interaktivnih grafikonov, animacij in spreminjanje grafičnih elementov brez osveževanja same spletne strani. Te lastnosti so ključne za vizualizacijo podatkov, še posebej v realnem času. Omogočata učinkovito izrisovanje na strani odjemalca, kar močno zmanjša potrebo po prenosu velikih grafičnih datotek s strežnika [30].

SVG: je vektorski grafični format, ki temelji na osnovi XML. Vsebina se definira znotraj značke SVG, podobno kot vsak drug dokument XML. Uporablja se za izris statičnih, dinamičnih, animiranih in interaktivnih vsebin. Elemente SVG je

možno stilirati s CSS in animirati z manipulacijo SVG DOM. Danes je podprt v vseh modernih brskalnikih na veliki paleti naprav. Zaradi široke podpore in neodvisnosti od ločljivosti (angl. resolution independence) je odlična izbira za grafično prikazovanje na različnih platformah [21, 30].

Canvas: HTML5 Canvas je vsestranski JavaScript API, ki prav tako kot SVG omogoča izris grafičnih vsebin. Za razliko od SVG, Canvas omogoča hitro izrisovanje rastrskih slik v 2D ali pa 3D, z uporabo spletne grafične knjižnice WebGL. Pri implementaciji je v dokumentu HTML potrebno definirati element <canvas>, ki ima vlogo vsebnika. Izrisovanje grafičnih vsebin v vsebnik <canvas> poteka z uporabo kode JavaScript [2, 30].

Katero izmed tehnologij je bolj smiselno uporabiti je odvisno od zahtev določene aplikacije. Obe tehnologiji imata svoje prednosti in slabosti, vendar sta vsesplošno deležni široke uporabe. Canvas je novejši kot SVG in v primerjavi z njim omogoča visoko zmogljivo risanje rastrskih grafičnih vsebin. Kadarkoli je potrebno izrisovanje na ravni slikovnih pik, je bolj primerna tehnologija Canvas. Prav tako je učinkovitejša od SVG, kadar je potrebno upodabljanje velikega števila grafičnih objektov. Nekatere izmed pomanjkljivosti tehnologije Canvas so slabe možnosti za upodabljanje besedil, odvisnost od ločljivosti in slabša učinkovitost pri upodabljanju na platno velikih dimenzij zaradi večjega števila izrisanih pikslov. SVG ima v primerjavi s Canvas nekatere prednosti, kot so npr. neodvisnost od ločljivosti, dobra podpora za animacije z uporabo deklarativne sintakse, neposredno stiliranje z uporabo CSS in podporo za rokovalnike dogodkov (angl. event handlers). Omogoča tudi neposredno manipulacijo elementov SVG s pomočjo JavaScript, na enak način kot poteka manipulacija elementov HTML. Tehnologiji SVG se pripisujejo očitki, da se pri povečani kompleksnosti upodabljanja zmanjša performančna učinkovitost. Ta aspekt se skozi čas izboljšuje, saj se ponudniki spletnih brskalnikov trudijo, da bi pohitrili zmožnosti upodabljanja s tehnologijo SVG [30].

V spletni aplikaciji, ki je bila razvita v okviru projekta SUNSEED, smo se odločili za visokofrekvenčno realnočasovno vizualizacijo s knjižnico Rickshaw.js,

ki temelji na tehnologiji SVG. Za vizualizacijo velikega števila preteklih podatkov smo uporabili knjižnico Dygraphs, ki temelji na tehnologiji Canvas. Za realnočasovni prikaz napetostnega profila smo se na podlagi vizualnih zahtev odločili za knjižnico Morris.js. Realnočasovni vektorski prikaz faznih kotov smo implementirali s pomočjo knjižnice Function Plot, ki je v osnovi namenjena vizualizaciji matematičnih funkcij. Za te knjižnice smo se odločili, saj predstavljajo dober kompromis med učinkovitostjo, funkcionalnostjo ter vizualno estetiko.

V nadaljevanju so z uporabo knjižnice Rickshaw.js predstavljeni primeri realnočasovne vizualizacije ter različni optimizacijski postopki, ki privedejo do boljše performančne učinkovitosti ob še vedno dovolj tekočem izrisovanju.

4.2 Realnočasovna vizualizacija

Realni čas je ohlapen izraz, ki v praksi zajema različne časovne intervale in frekvence izvedbe. Ustreznejši izraz bi bil "skoraj realni čas" (angl. near-real-time), saj so v praksi vedno prisotne latence oz. zakasnitve. Kadar so zahteve takšne, da se morajo podatki zbirati v realnem času, mora vsako orodje znotraj takega sistema delovati oz. nuditi svoje storitve v realnem času [33].

Osnovni potek realnočasovne vizualizacije v spletnem okolju sestoji iz naslednjih korakov:

- definiranje vsebnika za grafikon v strukturi dokumenta HTML;
- inicializacija grafikona z JavaScript (glede na dokumentacijo posamezne knjižnice);
- dodajanje in izris novih podatkovnih točk v grafikon.

V naslednjih primerih je intervalsko izrisovanje točk na grafikon simulirano z uporabo funkcij *setTimeout(funkcija, ms)* in *setInterval(funkcija, ms)*.

setTimeout(funkcija, ms)* in *setInterval(funkcija, ms): Funkciji, ki ju omogoča okenski objekt (angl. window object), ki je ponavadi kar brskalnik, se uporabljata pri izvajanju kode v določenih časovnih intervalih. Funkciji sprejmeta

dva argumenta, pri čemer je prvi argument funkcija, ki se naj izvede, drugi argument pa je časovni zamik v milisekundah. Funkcija *setTimeout(funkcija, ms)* izvede definirano funkcijo *funkcija* po preteku določenega števila milisekund, *ms*, enkrat, *setInterval(funkcija, ms)* pa redno izvaja definirano funkcijo *funkcija* s fiksnim časovnim zamikom, *ms*, med vsakim klicem, do preklica izvajanja s funkcijo *clearInterval(intervalID)*, kjer *intervalID* predstavlja identifikacijo, ki jo vrneta *setTimeout(funkcija, ms)* oziroma *setInterval(funkcija, ms)* [23].

V nadaljevanju se podrobneje osredotočamo v samo implementacijo. Predstavljeni so različni primeri realnočasovne vizualizacije z uporabo knjižnice Rickshaw.js.

4.2.1 Osnovni primer

Simulacija vizualizacije z enosekundnim intervalom osveževanja: Začetni cilj je izvesti simulirano vizualizacijo časovne vrste s črtnim grafikonom, ki v danem trenutku vsebuje največ sto podatkov in se v realnem času posodablja z enosekundnim časovnim intervalom.

Začnemo s kreiranjem strukture dokumenta HTML, kjer opredelimo gradnike , ki so vsebniki inicializiranemu grafikonu ter vertikalni osi. Primer vsebnika HTML je ponazorjen v kodnem izseku 4.1.

Kodni izsek 4.1: Osnovni vsebnik HTML.

```
1 <body>
2     <div id="chart_container">
3         <div id="y_axis"></div>
4         <div id="demo_chart"></div>
5     </div>
6
7     // ...
8 </body>
```

V naslednjem koraku je potrebna inicializacija grafikona Rickshaw.js. V datoteki JavaScript moramo najprej definirati dve spremenljivki. Prva predstavlja interval posodabljanja, druga pa hrani trenutno meritev, ki se vstavi v grafikon. Definirati je potrebno še spremenljivko za grafikon in ji določiti ciljni vsebnik, tip, dimenzije, stil, maksimalno dovoljeno število podatkov in druge lastnosti. Knjižnica Rickshaw.js privzeto ne izrisuje koordinatnih osi, zato v ta namen definiramo še dodatno spremenljivko, ki inicializira vertikalno os. Implementacija je prikazana v kodnem izseku 4.2.

Kodni izsek 4.2: Inicializacija Rickshaw.js črtnega grafikona.

```
1  var updateInterval = 1000; // 1000 ms
2  var tmpData = {};
3
4  /* Rickshaw.js chart initialization */
5  var chart1 = new Rickshaw.Graph({
6      element: document.querySelector("#demo_chart"),
7      width: "500",
8      height: "200",
9      renderer: "line",
10     min: "0",
11     max: "70",
12     series: new Rickshaw.Series.FixedDuration([{
13         name: 'datapoint',
14         color: '#EC644B'
15     }], undefined, {
16         timeInterval: updateInterval,
17         maxDataPoints: 100
18     })
19 });
20
21 /* Y-axis initialization */
22 var y_axis = new Rickshaw.Graph.Axis.Y({
23     graph: chart1,
24     orientation: 'left',
25     tickFormat: function (y) {
```

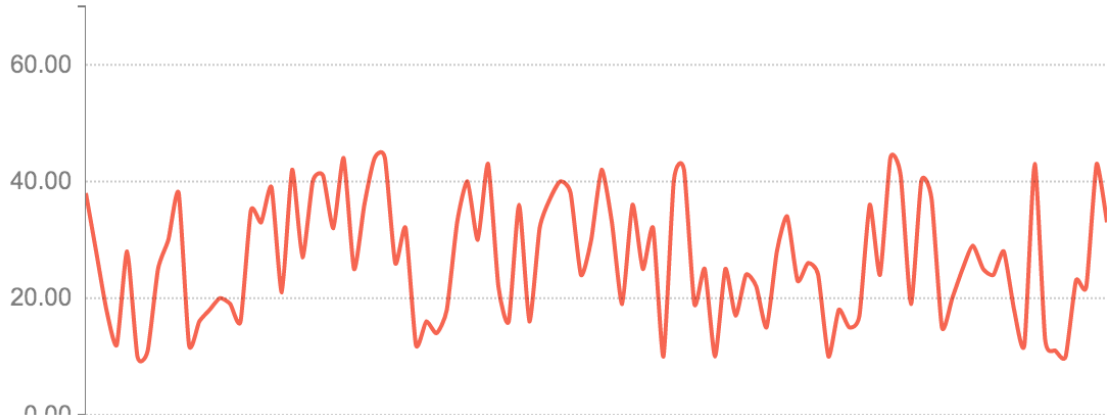
```
26          return y.toFixed(2);
27      },
28      ticks: 5,
29      element: document.getElementById('y_axis'),
30  });
31
32  /* ... */
```

V tej fazi grafikon še ni izrisan. Potrebno je dodati še kodo, ki generira naključna števila in jih v vsakem časovnem intervalu vstavi v grafikon. Za lažjo simulacijo podatkovnega toka (angl. data stream) ustvarimo funkcijo *insertRandomDatapoints()*, ki v vsakem časovnem intervalu generira naključno število, ga v ustreznem formatu vstavi v spremenljivko *tmpData* in jo nato vstavi v grafikonovo trenutno serijo podatkov. Na koncu funkcije se s klicem *render()* opravi izris grafikona. Celotna funkcija je definirana znotraj časovne funkcije *setInterval(funkcija, ms)*, ki izvede prej omenjeno funkcijo v vsaki iteraciji, ki se izvede na definiran časovni interval. Implementirana koda je predstavljena v kodnem izseku 4.3. Slika 4.1 prikazuje tako zgeneriran grafikon.

Kodni izsek 4.3: Dodajanje podatkovnih točk ter intervalski izris grafikona.

```
32  /* ... */
33
34  /* Timer function that calls @insertRandomDatapoints(arg) every
        @updateInterval milliseconds */
35  setInterval(function () {
36      insertRandomDatapoints();
37  }, updateInterval);
38
39
40  /* Function that generates and inserts random datapoint into
       the chart */
41  function insertRandomDatapoints() {
42      tmpData = {
```

```
43            datapoint: Math.floor(Math.random() * 35) + 10
44        };
45
46        chart1.series.addData(tmpData);
47        chart1.render();
48    }
```



Slika 4.1: Realnočasovni grafikon, ki se posodablja vsako sekundo. V danem trenutku vsebuje največ sto meritev. Na sliki je predstavljeno stanje grafikona po več kot sto sekundah upodabljanja meritev.

Takšen način implementacije je učinkovit na enostavnih primerih, kjer je časovni interval posodabljanja relativno dolg. Povečana težavnost se pojavi pri vizualizaciji visokofrekvenčnega toka podatkov, saj je večja verjetnost za pojav ozkih grl (angl. bottlenecks) bodisi v omejitvah brskalnikov bodisi v omejitvah strojne opreme.

4.2.2 Visokofrekvenčni prikaz v realnem času

Funkcija *setInterval(funkcija, ms)* v osnovi ni procesorsko intenzivna, lahko pa postane, kadar se sočasno izvaja veliko število intervalov ali pa ko se ti intervali izvajajo v zelo kratkih ciklih. V primeru, ko je potrebno vizualizirati podatkovni tok, kjer je frekvenca podatkov visoka (npr. 50 Hz), bi bilo potrebno grafikon izrisovati v 20 ms intervalih. Takšen interval se v praksi z uporabo funkcije *setIn-*

terval(funkcija, ms) zaradi omejitev brskalnikov ter strojne opreme izkaže za nezanesljivega in procesorsko precej zahtevnega. Možne pohitritve lahko dosežemo z modifikacijami klica funkcije *setInterval(funkcija, ms)*, pri čemer ustvarimo kompromis, kot je prikazan v nadaljevanju. Klic funkcije *setInterval(funkcija, ms)* lahko optimiziramo na takšen način, da povečamo interval v milisekundah, pri čemer pa v izrisovalni funkciji *funkcija* ustvarimo zanko, ki znotraj enega intervala namesto ene točke vstavi večje število točk, ki jih na grafikon upodobi z enim klicem funkcije *render()*.

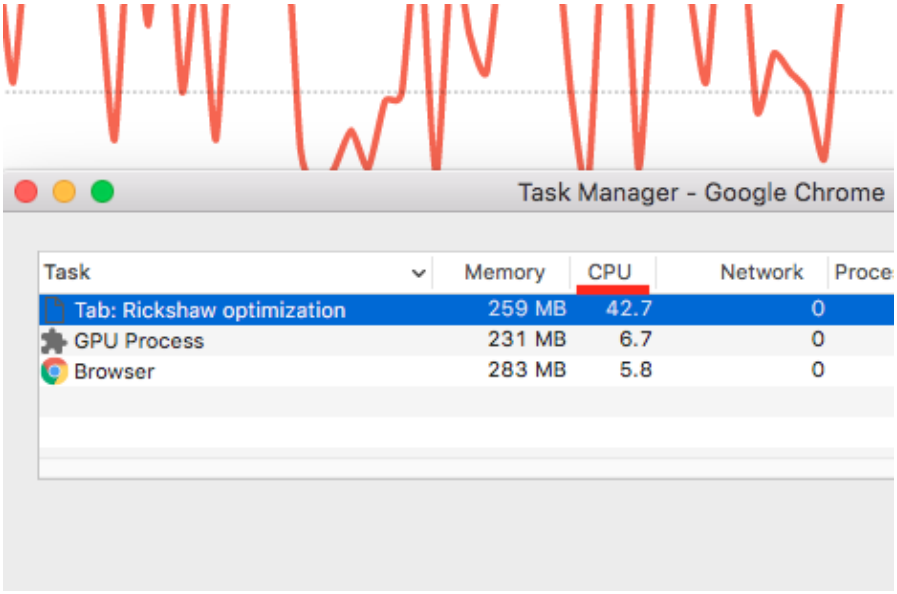
Simulacija visokofrekvenčne vizualizacije: Izrisovati želimo podatkovni tok s frekvenco 50 Hz. Uporabljeno je osnovno ogrodje iz sekcije 4.2.1. Časovni interval znotraj funkcije *setInterval(funkcija, ms)* nastavimo na 100 ms, kot je prikazano v vrstici 1 kodnega izseka 4.4. Da dosežemo frekvenco 50 Hz, moramo na grafikon izrisovati pet meritev znotraj enega časovnega intervala. To implementiramo s pomočjo zanke, kot je prikazano v vrstici 16 kodnega izseka 4.4. Po zaključku zanke se na novo vstavljene meritve izrišejo v grafikon.

Kodni izsek 4.4: Primer optimizacije realnočasovne vizualizacije z visoko frekvenco.

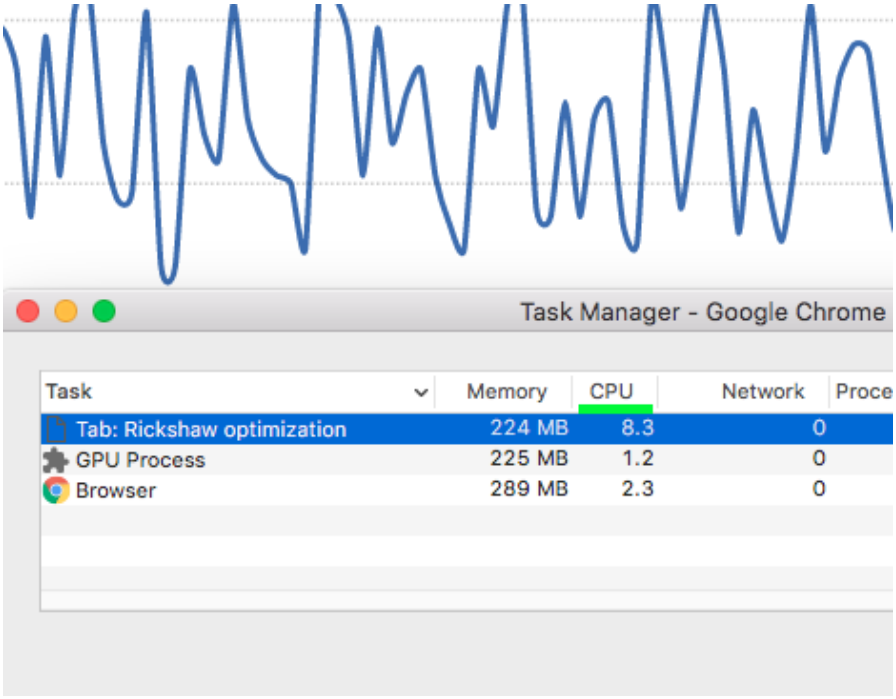
```
1  var updateInterval = 100; // changed to 100 ms
2  var tmpData = {};
3
4  /* ...Rickshaw.js chart initialization... */
5
6  /* Timer function that calls @insertRandomDatapoints(arg) every
       @updateInterval milliseconds */
7  setInterval(function () {
8      insertRandomDatapoints();
9  }, updateInterval);
10
11
12 /* Function that generates and inserts random datapoints into
      the chart */
13 function insertRandomDatapoints() {
```

```
14
15    /* Add multiple points to chart */
16    for (var i = 0; i < 5; i++) {
17        tmpData = {
18            datapoint: Math.floor(Math.random() * 35) + 10
19        };
20        chart1.series.addData(tmpData);
21    }
22      chart1.render();
23  }
```

S tem načinom pohitritve smo zmanjšali število funkcijskih klicev in tako dosegli večjo učinkovitost v zameno za vizualno manj tekočo animacijo. Primerjava obremenitve procesorja pred in po optimizaciji je prikazana na sliki 4.2. Opazimo, da se je obremenjenost procesorja po optimizaciji (4.2b) izrazito zmanjšala v primerjavi z obremenjenostjo pred optimizacijo (4.2a). Potrebno se je zavedati, da funkcija *setInterval(funkcija, ms)* v osnovi ni mišljena kot nadomestilo za natančno uro, saj je točnost časovnih intervalov odvisna od trenutne (ne)obremenjenosti virov in od stanja predhodnega izrisa. Zaradi teh razlogov je potrebno v praksi uporabiti še številne dodatne mehanizme, ki preprečujejo neizrisovanje meritev zaradi morebitne prevelike počasnosti izrisovanja trenutnih podatkov. Pohitritve so možne tudi z uporabo knjižnic, ki temeljijo na tehnologiji Canvas, vendar pa je potrebno upoštevati tudi ostale omejitve tega pristopa.



(a)



(b)

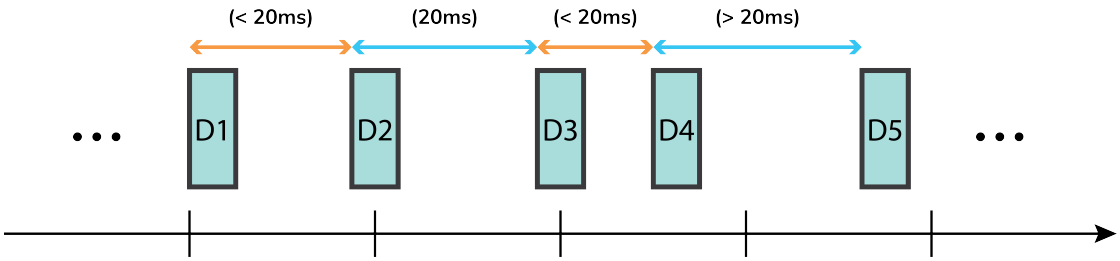
Slika 4.2: Slika prikazuje primerjavo uporabe CPE v Google Chrome[a] upravitelju opravil (angl. task manager) med izvajanjem realnočasovne vizualizacije. (a) Uporabljenost CPE za izris grafikona pred optimizacijo (20 ms interval izrisovanja). (b) Uporabljenost CPE za izris grafikona po pohitritvi, kot je prikazano v kodnem izseku 4.4.

[a] `https://www.google.com/chrome/`

Visokofrekvenčna vizualizacija podatkovnega toka: V resničnih situacijah podatkovni tok izvira iz zunanjih virov. V nadaljevanju želimo vizualizirati podatkovni tok s povprečno frekvenco 50 Hz. Meritve senzorjev se iz naprav IoT posredujejo na teme MQTT v formatu JSON in vsebujejo enolično informacijo o identifikatorju naprave, zaporedno številko poročila (v vsaki sekundi zavzema vrednost od 0-49) in številsko vrednost meritve (npr. magnituda), kot je prikazano v kodnem izseku 4.5. Slika 4.3 prikazuje primer latence med podatki v visokofrekvenčnem podatkovnem toku.

Kodni izsek 4.5: Primer prejete meritve v formatu JSON.

```
1 {
2   "node_id": "167002045410006104c5a000a00000f5",
3   "report_num": 0,
4   "v1": 235.14799264
5 }
```



Slika 4.3: Primer latence med prejetimi podatki v podatkovnem toku s povprečno frekvenco 50 Hz.

V strežniku Node.js se z uporabo knjižnice MQTT.js avtenticiramo in naročimo na ustrezno temo MQTT, na katero pošiljatelj objavlja meritve. Implementirati je potrebno funkcijo, ki deluje kot dogodkovni poslušalec (angl. event listener) in se izvede vsakič, ko prejmemo meritev iz naročene teme. Znotraj dogodkovnega poslušalca je potrebno meritev v formatu JSON razčleniti (angl. parse). Z uporabo knjižnice Socket.io posredujemo razčlenjeno meritev v naprej predvideno

sobo (angl. room), na katero je prijavljen odjemalec. Osnovna implementacija opisanega postopka je razvidna v kodnem izseku 4.6.

Kodni izsek 4.6: Komunikacija strežnika Node.js z odjemalcem.

```
1 /* Parse and forward MQTT message to the client */
2 client.on('message', function(topic, message) {
3   parsedData = JSON.parse(message);
4   io.in(topic).emit('realtimeData', parsedData);
5 }
```

Na odjemalčevi strani inicializiramo grafikon Rickshaw.js, kot je prikazano v kodnem izseku 4.2. S knjižnico Socket.io se prijavimo v ustrezno sobo in definiramo poslušalca, ki izvede definirano kodo vsakič, kadar je prejeta nova meritev. V teoriji bi lahko vsakič, ko prejmemo novo meritev, izrisali le to v grafikon. Ta način postane zelo neučinkovit pri visokofrekvenčni vizualizaciji, kot v tem primeru, ko se prejema povprečno petdeset meritev na sekundo. Takšna implementacija (kodni izsek 4.7) bi zahtevala, da se grafikon izrisuje v povprečju petdesetkrat na sekundo oz. v 20 ms intervalih, kar pri brskalniku izzove performančne težave.

Kodni izsek 4.7: Primer neučinkovite visokofrekvenčne vizualizacije.

```
1  var chartData = {};
2
3  /* Update chart when new data is received */
4  socket.on('realtimeData', function (message) {
5    chartData = {
6      datapoint: message["v1"]
7    }
8    chart1.series.addData(chartData);
9    chart1.render();
10 }
```

V praksi lahko boljšo učinkovitost dosežemo z enostavno optimizacijo. Podobno, kot je opisano v razdelku 4.2.2, implementiramo algoritem, ki namesto ob vsaki meritvi izriše grafikon po vsaki peti prejeti meritvi. S tem v teoriji zmanjšamo interval izrisovanja iz 20 ms na 100 ms, kar se močno pozna v učinkovitosti vizualizacije. Osnovna implementacija takšne optimizacije je predstavljena v kodnem izseku 4.8.

Kodni izsek 4.8: Primer optimizirane visokofrekvenčne vizualizacije.

```
1  var chartData = {};
2  var dataIndex = 1;
3
4  /* Update chart when new data is received */
5  socket.on('realtimeData', function (message) {
6    chartData = {
7      datapoint: message["v1"]
8    }
9    chart1.series.addData(chartData);
10
11   /* Render chart for every 5 added data points */
12   if (dataIndex % 5 == 0) {
13     chart1.render();
14   }
15
16   /* Update index */
17   if (dataIndex < 50) {
18     dataIndex++;
19   } else {
20     dataIndex = 1;
21   }
22 }
```

Prikazan je eden izmed načinov učinkovite vizualizacije visokofrekvenčnega podatkovnega toka v realnem času. V praksi se pokaže, da v visokofrekvenčnih podatkovnih tokih nastopajo različne latence med sprejetimi podatki, kot je prikazano

na sliki 4.3. Te latence lahko zaradi neenakomernih intervalov med prihodi podatkov privedejo do manj tekočega izrisovanja realnočasovnih podatkovnih tokov. Kadar vizualizacija ni strogo časovno kritična, lahko v takšnih primerih za vizualno bolj tekočo vizualizacijo uporabimo zakasnjeno upodabljanje. Pri realnočasovnih aplikacijah, kjer je sprejemljiva nekaj sekundna latenca, lahko nove podatke shranjujemo v vrsto (angl. queue) in jih zakasnjeno upodobimo. Pri takšnem načinu upodabljanja lahko za bolj tekočo vizualizacijo uporabimo časovno funkcijo *setInterval(funkcija, ms)* z ustreznimi optimizacijami, kot so opisane v kodnem izseku 4.4.

Poglavje 5

Vizualizacija distribucijskega električnega omrežja

Končna aplikacija je predstavljena v obliki odzivnega grafičnega vmesnika. Omogoča učinkovit vpogled v trenutno in preteklo stanje distribucijskih električnih omrežij. Uporabnik lahko izbira med različnimi podomrežji, ki se v interaktivnem zemljevidu nato upodobijo v obliki vozlišč in povezav. Uporabnik lahko z akcijami na zemljevidu izbira med različnimi realnočasovnimi vizualizacijami podatkovnih tokov PMU in DSSE. Aplikacija omogoča tudi učinkovito vizualizacijo velike množice preteklih meritev. V nadaljevanju je podrobneje opisan način, s katerim smo različne gradnike združili v homogeno aplikacijo, ki učinkovito služi namenu opazovanja distribucijskega električnega omrežja.

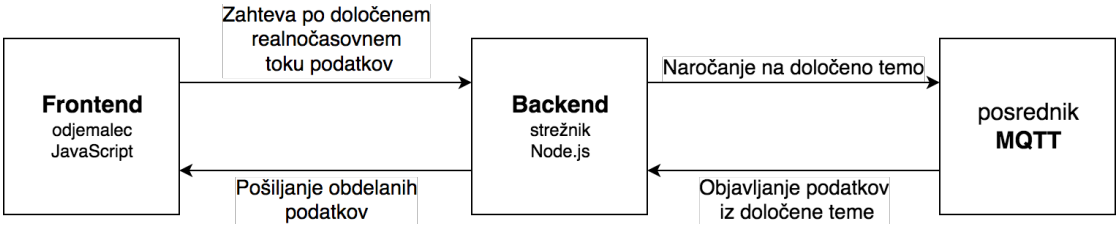
5.1 Komunikacija strežnik-odjemalec

Realnočasovna komunikacija strežnika Node.js z odjemalcem poteka s pomočjo knjižnice Socket.io.

Osnovni potek komunikacije (slika 5.1):

- odjemalec pošlje zahtevo strežniku za specifičen realnočasovni podatkovni tok;

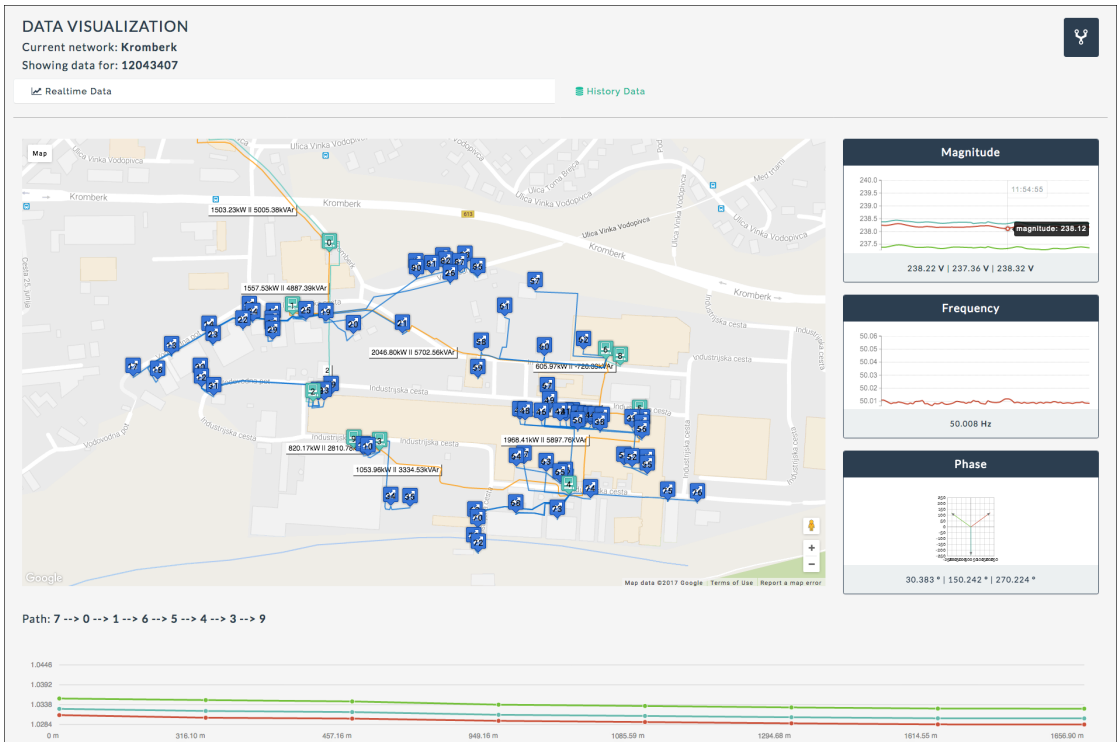
- strežnik se naroči na zahtevano temo MQTT po vzorcu objavi-naroči;
- posrednik MQTT posreduje strežniku realnočasovni podatkovni tok;
- strežnik razčleni ter obdela podatke;
- strežnik pošlje obdelane realnočasovne podatke odjemalcu preko spletnih vtičnic;
- odjemalec upodobi vizualizacijo v realnem času.



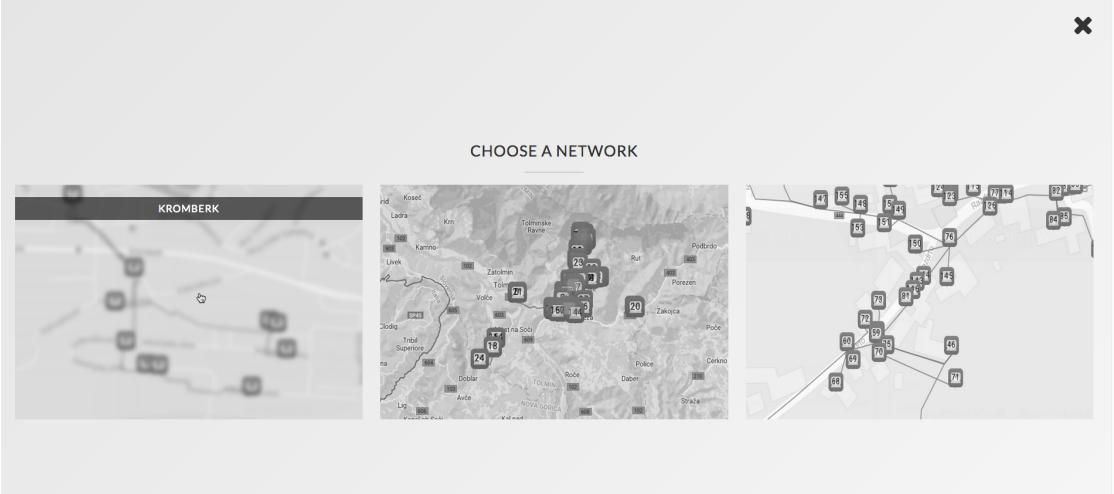
Slika 5.1: Pregled osnovnega poteka komunikacije med odjemalcem, strežnikom in posrednikom MQTT.

5.2 Grafični vmesnik in gradniki aplikacije

Aplikacija je predstavljena kot pregledna plošča z vizualno čistim uporabniškim vmesnikom (slika 5.2). Uporabniku omogoča izbiro med različnimi distribucijskimi električnimi podomrežji (slika 5.3), podaja pregled nad realnočasovnimi meritvami različnih vozlišč ter omogoča vizualizacijo velikega števila preteklih meritev iz podatkovne baze.



Slika 5.2: Pregledna plošča oz. grafični uporabniški vmesnik spletne aplikacije za vizualizacijo podatkov.



Slika 5.3: Prikaz možnosti izbire uporabnika med različnimi omrežji.

5.2.1 Interaktivni Google Maps zemljevid

Osrednji povezovalni grafični element aplikacije je interaktivni zemljevid Google Maps, ki prikazuje vsa vozlišča izbranega omrežja ter povezave med njimi. Interaktivni zemljevid omogoča:

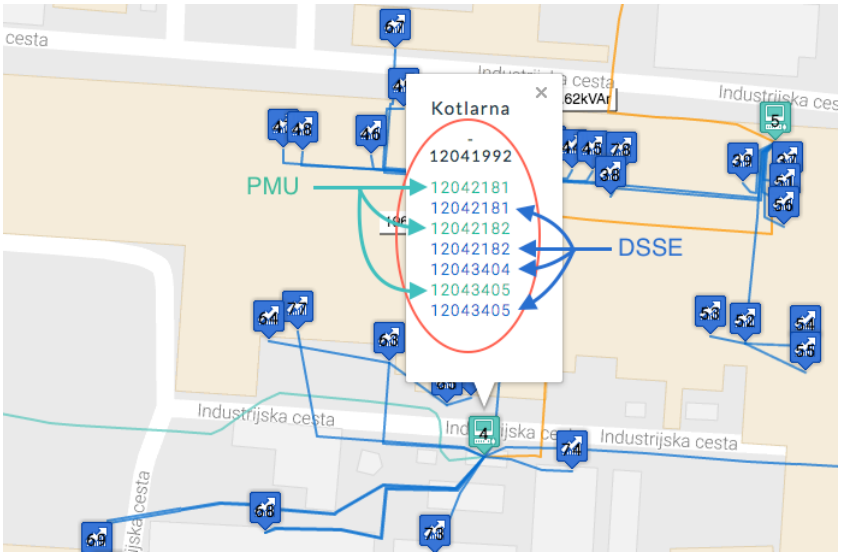
- izbiro vizualizacije podatkovnega toka specifičnega vozlišča v omrežju;
- izbiro med vizualizacijo podatkov DSSE ali PMU izbranega vozlišča;
- realnočasovni vpogled v pretok moči v obliki tekstovnih oznak na zemljevidu.

Za izris geografsko pravilnih lokacij vozlišč in povezav na zemljevid smo podatke o lokacijah morali pretvoriti v ustrezen format. Geografski podatki so bili izvlečeni iz datotek AutoCAD[1] in so bili podani v koordinatnem sistemu Gauss-Krüger. S pomočjo različnih skript smo, za lažjo uporabo z zemljevidom Google, informacije o lokacijah pretvorili v koordinatni sistem WGS84 in podatke shranili v datoteko JSON, kot je prikazano v kodnem izseku 5.1. Vozlišča in povezave podomrežij so izrisani s pomočjo zanke, ki obhodi geografske podatke iz zunanje datoteke JSON. V vsaki iteraciji zanke se v zemljevid dodajo označevalci (angl. markers), na katere so vezani dogodkovni poslušalci, ki ob kliku odprejo informacijsko okno (angl. info window). Znotraj informacijskega okna se izpišejo različni podatkovni toki tega vozlišča, kot je prikazano na sliki 5.4. S klikom na enega izmed njih se proži vizualizacija ustreznega podatkovnega toka. Ob povezavah med vozlišči je v obliki tekstovnih oznak prikazan pretok moči, ki se osvežuje vsako sekundo.

[1] `https://www.autodesk.com/products/autocad/overview`

Kodni izsek 5.1: Primer geografskih podatkov in ostalih informacij posameznega vozlišča podomrežja v formatu JSON.

```
{
  "lnodes": {
    "coordinates": [{
      "lat": 12.3456789123456,
      "lng": 12.3456789123456,
      "id": 12345678,
      "name": "Jogi",
      "points": [{
        "pointId": 23456789,
        "pointName": "SN.Jogi",
        "status_dsse": "active"
      }],
      "status_pmu": "active"
    }]
  }
}
```

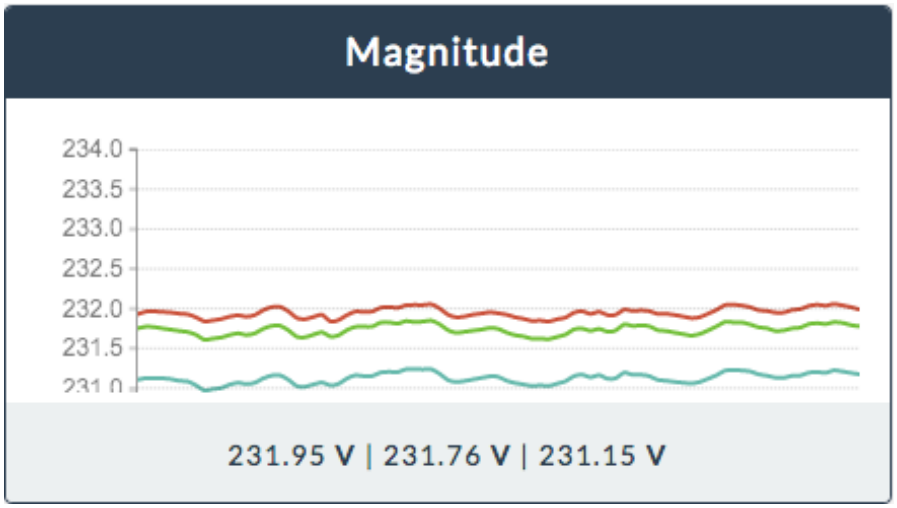


Slika 5.4: Prikaz funkcionalnosti, kjer lahko uporabniki s klikom na vozlišče izbirajo med prikazovanjem meritev PMU in izračunov DSSE. Meritve PMU so označene z zelenkastimi puščicami, medtem ko so izračuni DSSE označeni z modrimi puščicami.

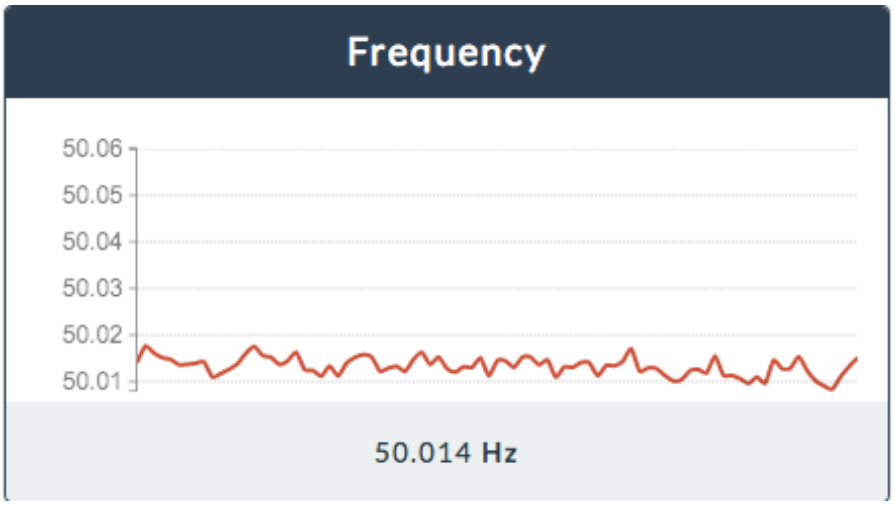
5.2.2 Realnočasovni grafikoni

Realnočasovni grafikoni so jedro aplikacije in se uporabljajo pri vizualizaciji meritev specifičnih podatkovnih tokov enot PMU. Prejeti podatki so v formatu JSON in vsebujejo informacije, kot so trenutne meritve magnitud, frekvenc in faznih kotov ter ostale informacije, ki so specifične za vsako vozlišče (slika 2.2). Frekvenca podatkovnih tokov enot PMU je 50 Hz. Realnočasovna vizualizacija v aplikaciji sestoji iz sledečih gradnikov:

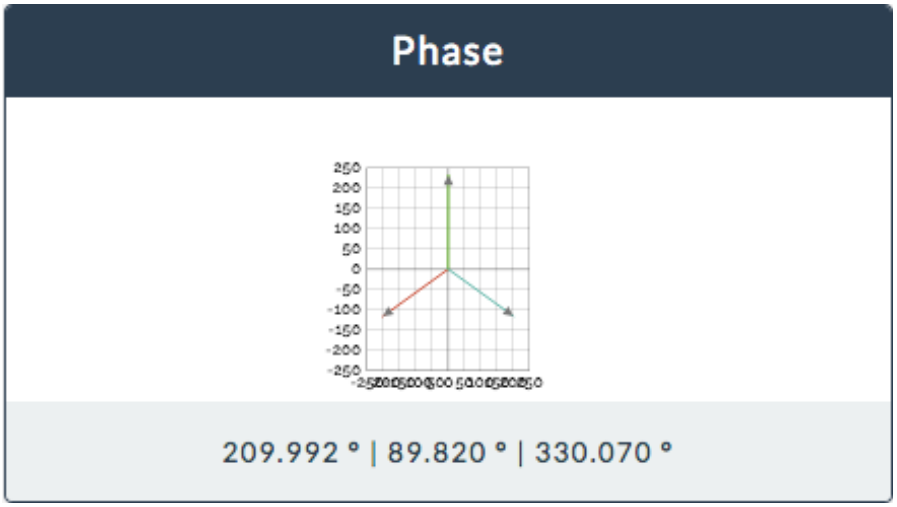
- grafikon magnitud (črtni grafikon z več nizi), ki je prikazan na sliki 5.5;
- grafikon frekvence (črtni grafikon), ki je prikazan na sliki 5.6;
- vektorski prikaz faznih kotov, ki je prikazan na sliki 5.7.



Slika 5.5: Dvo-sekundna vizualizacija magnitud s črtnim grafikonom z več nizi (angl. multiple series line chart). Na osi y so prikazane vrednosti magnitud v Voltih (V).



Slika 5.6: Dvo-sekundna vizualizacija frekvenc s črtnim grafikonom (angl. line chart). Na osi y so prikazane frekvence v Hertzih (Hz).

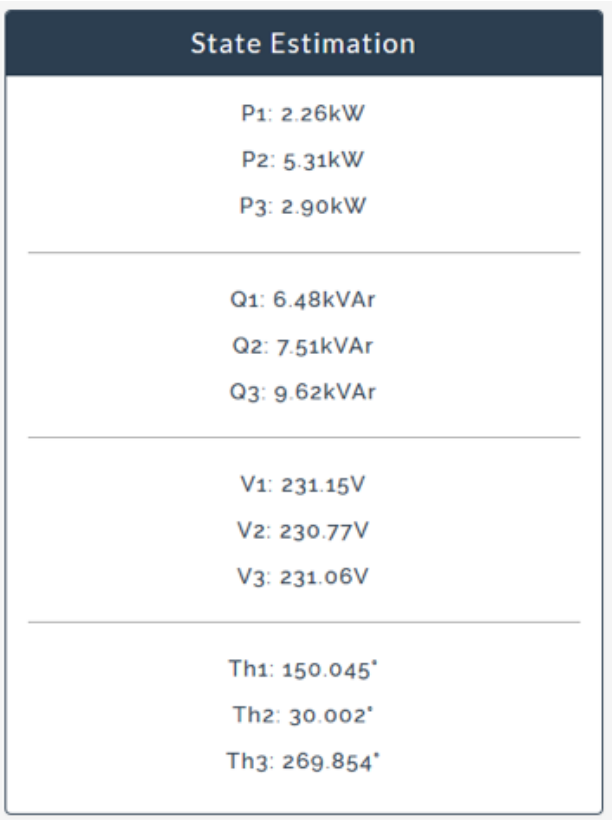


Slika 5.7: Vektorski prikaz faznih kotov z uporabo knjižnice Function Plot. Koordinatni osi prikazujeta napetosti v Voltih (V), medtem ko dolžine vektorjev ponazarjajo amplitudo napetosti.

Za realnočasovni prikaz meritev s črtnim grafikonom je bila uporabljena knjižnica Rickshaw.js, za vektorsko ponazoritev pa Function Plot. Ti knjižnici predstavljata dober kompromis med funkcionalnostjo ter učinkovitostjo. Realnočasovno upodabljanje s črtnim grafikonom je bilo optimizirano s pristopi, ki so podrobneje opisani v sekciji 4.2.2. Rezultat je vizualno prijetna in tekoča animacija črtnega grafikona s konstantnim gibanjem meritev. Črtna grafikona magnitud in frekvence vsebujeta v vsakem trenutku največ sto točk na en podatkovni niz (angl. series).

5.2.3 Realnočasovni prikaz podatkov ocenjevalnika stanja (DSSE)

Uporabnik lahko za vsako vozlišče v omrežju v informacijskem oknu izbira med različnimi podatkovnimi toki izračunov DSSE. Izračuni se v realnem času osvežujejo enkrat na sekundo in so prikazani v obliki besedila, kot je prikazano na sliki 5.8. V tem prikazu so zajete delovne moči, jalove moči, napetosti in fazni koti.



Slika 5.8: Izračuni DSSE, ki se osvežujejo vsako sekundo. Prikazani so izračuni za delovno moč (P1-P3), jalovo moč (Q1-Q3), napetost (V1-V3) in fazne kote (Th1-Th3).

5.2.4 Grafični prikaz napetostnega profila

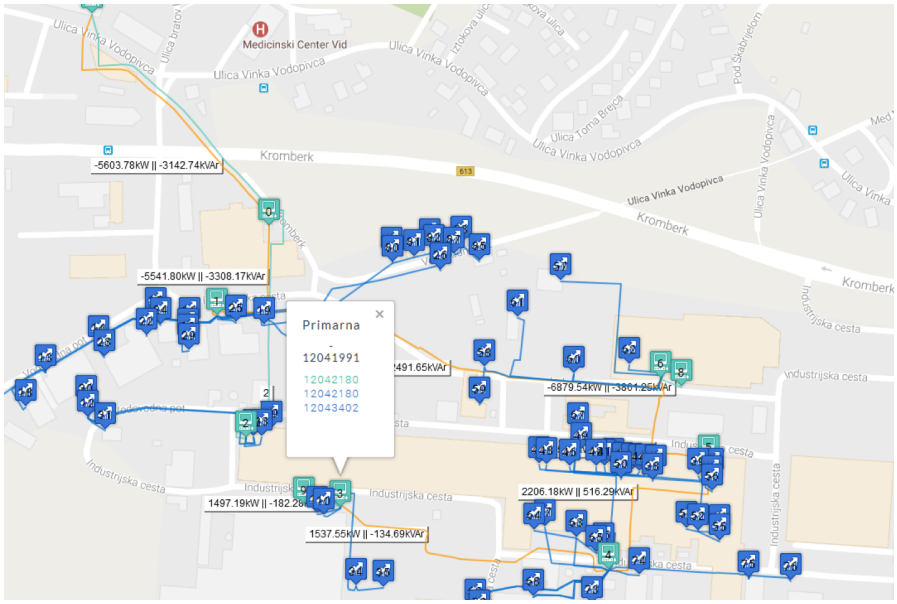
Napetostni profil je vizualiziran s črtnim grafikonom z več nizi, čigar vertikalna os prikazuje normirane magnitude, horizontalna pa razdalje vozlišč od začetnega do izbranega vozlišča (v metrih). Prikaz napetostnega profila sestoji iz:

- iskanja poti z rekurzivnim algoritmom po incidenčni matriki, ki prikazuje povezave med vozlišči;
- računanja razdalj med vozlišči s klici Google Maps API;
- realnočasovne vizualizacije napetostnega profila s črtnim grafikonom z več nizi.

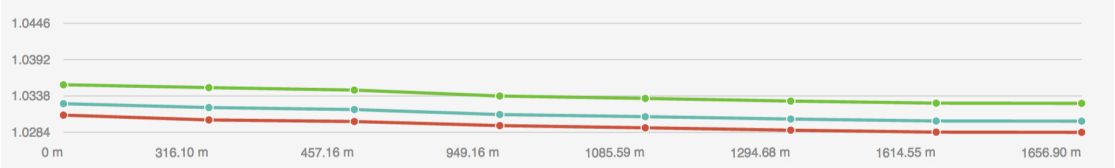
Incidenčno matriko smo za lažjo implementacijo hranili v formatu JSON, kot je predstavljeno v kodnem izseku 5.2. Ko uporabnik klikne na vozlišče v omrežju, se požene algoritem, ki izvede zgoraj naštete postopke. Po kliku se oranžno obarva celotna pot preko vseh vozlišč od začetnega do izbranega vozlišča (slika 5.9). Uporabniku se prikaže realnočasovni črtni grafikon z vsemi vmesnimi meritvami od začetnega do izbranega vozlišča in se osvežuje vsako sekundo (slika 5.10).

Kodni izsek 5.2: Incidenčna matrika povezav med vozlišči.

```
{
  "iMatrixKromberk": [
    [1, 0, 0, -1, 0, 0, 0, 0, 0],
    [-1, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, -1, 0, 0, 0, 1],
    [0, 1, 0, 0, 0, 0, 0, -1, 0],
    [0, -1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, -1, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, -1, 1, 0, 0],
    [0, 0, 0, 1, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, -1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, -1]
  ]
}
```



Slika 5.9: Po uporabnikovem kliku na eno izmed osrednjih vozlišč se požene algoritem za iskanje poti, ki izbere in obarva najdeno pot ter izračuna vse geografske razdalje med vozlišči.

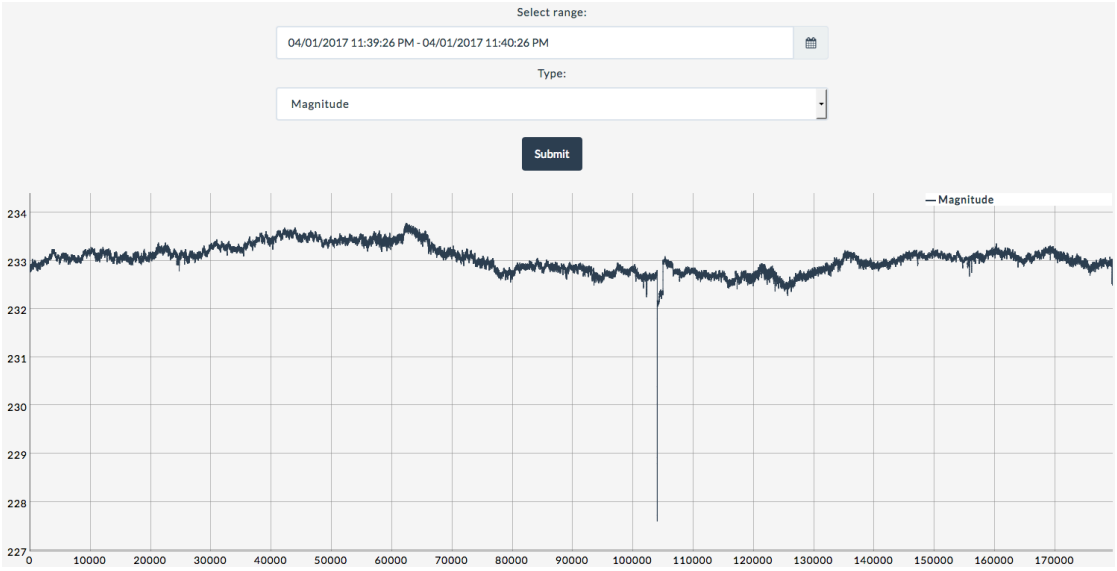


Slika 5.10: Podatki napetostnega profila so prikazani kot črtni grafikon z več nizi, ki se osvežuje vsako sekundo. Na osi x so prikazane geografske razdalje med vozlišči v metrih (m), na osi y pa normirane vrednosti magnitud v Voltih (V). Grafikon je implementiran s knjižnico Morris.js.

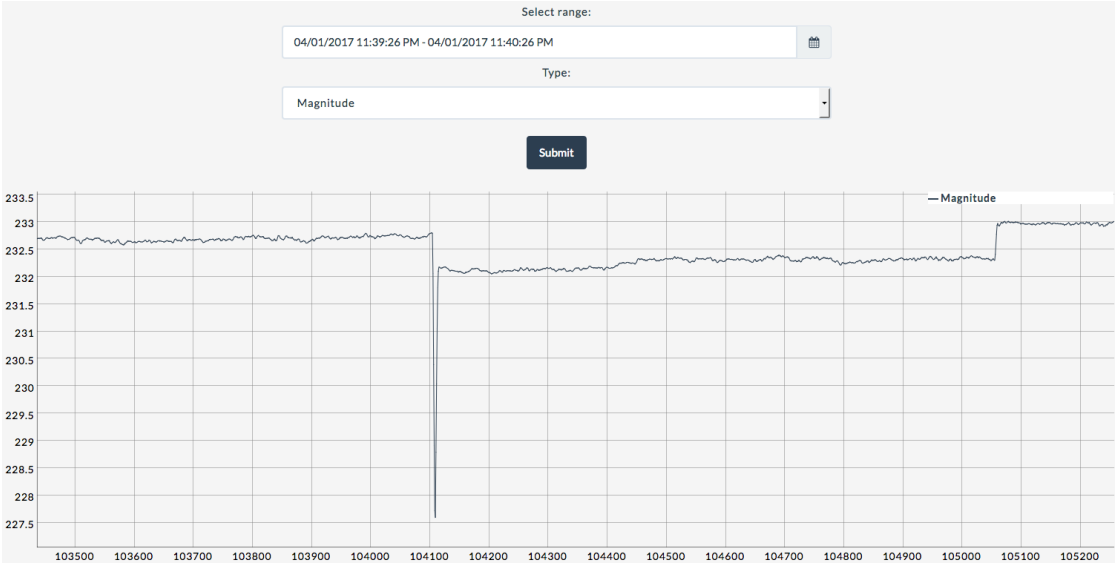
5.2.5 Vizualizacija preteklih meritev

Pretekle meritve se hranijo v podatkovni bazi MongoDB. Uporabnik v zavihku "Pretekle meritve" izbere začetni in končni datum meritev ter želeno količino (npr. magnitudo), kar prikazuje slika 5.11. Po potrditvi intervala se meritve vizualizirajo v črtni grafikon, ki omogoča povečavo (slika 5.11b prikazuje povečan grafikon

meritev s slike 5.11a) in s tem lažjo identifikacijo anomalij v meritvah.



(a)



(b)

Slika 5.11: Vizualizacija velikega števila preteklih meritev. (a) Grafikon z vsemi meritvam znotraj izbranega intervala. (b) Povečan grafikon, ki prispeva k lažji identifikaciji anomalij v omrežju. Os x predstavlja zaporedno številko meritve, os y pa magnitudo v Voltih (V).

Poglavje 6

Sklepne ugotovitve

V okviru diplomskega dela smo razvili spletno aplikacijo za vizualizacijo distribucijskih električnih omrežij. Osrednji povezovalni element aplikacije je interaktivni zemljevid, ki ponazarja geografsko natančen model izbranega podomrežja. Razvita aplikacija omogoča realnočasovno vizualizacijo visokofrekvenčnih podatkovnih tokov enot PMU, realnočasovno vizualizacijo napetostnega profila in izračunov DSSE ter vizualizacijo velikega števila preteklih meritev. Poleg tega smo predstavili tudi motivacijo in teoretično ozadje distribucijskih električnih omrežij. Prikazali smo različne tehnologije, knjižnice in vizualizacijske pristope, ki se lahko v praksi uporabijo za izgradnjo realnočasovnih spletnih aplikacij, katerih namen je lažje opazovanje stanj distribucijskih električnih omrežij s pomočjo različnih vizualizacij. Prikazali smo tudi enostavne optimizacijske postopke, ki se lahko uporabijo pri visokofrekvenčni realnočasovni vizualizaciji v spletnem okolju. Na koncu smo predstavili celotno aplikacijo in podrobneje opisali posamezne gradnike, njihovo nalogo in način implementacije. Končna aplikacija je bila za namene projekta SUNSEED uspešno preizkušena tudi v produkcijskem okolju.

Uporabljeni pristopi in načini izgradnje aplikacije niso omejeni samo na področje distribucijskih električnih omrežij. Ideje tega diplomskega dela se lahko prenesejo na številne problemske domene, kjer ima realnočasovna vizualizacija podatkovnih tokov velik pomen (npr. internet stvari).

6.1 Nadaljnje delo

Ugotovitve diplomskega dela lahko služijo kot osnova za številne nadaljnje izboljšave. Upravičeno lahko v prihodnosti pričakujemo, da bo s kontinuiranim razvojem in prihodom pametnih električnih omrežij opazovanje stanj sistemov v realnem času pridobilo še večji pomen. V prihodnosti bodo lahko v ta namen razvite aplikacije glavna podpora odločanju in bodo nadgrajene še z veliko novimi funkcionalnostmi.

Razvito aplikacijo bi lahko v prihodnje nadgradili še z naslednjimi funkcionalnostmi:

- avtomatsko odkrivanje nepravilnosti glede na realnočasovni podatkovni tok;
- avtomatsko generiranje alarmov na podlagi odkritih nepravilnosti;
- avtomatsko odpravljanje nepravilnosti;
- prikazovanje trendov;
- napovedovanje trendov na podlagi preteklih meritev in izračunov, ki so na voljo v podatkovni bazi.

Ponudniki spletnih brskalnikov prav tako nenehno izboljšujejo funkcionalnosti ter performančno učinkovitost brskalnikov. To bo v prihodnosti omogočalo lažji razvoj realnočasovnih spletnih aplikacij za vizualizacijo visokofrekvenčnih podatkovnih tokov.

Literatura

[1] Bootstrap; The most popular HTML, CSS, and JS library in the world. http://getbootstrap.com/. Accessed: 2017-10-13.

[2] Canvas API - Web APIs — MDN. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. Accessed: 2017-10-18.

[3] Dygraphs. hhttp://dygraphs.com/. Accessed: 2017-06-08.

[4] Electricity Distribution - IER. https://www.instituteforenergyresearch.org/electricity-distribution/. Accessed: 2017-09-27.

[5] Express.js. https://expressjs.com. Accessed: 2017-06-08.

[6] Function Plot - 2d function plotter powered by d3. https://mauriciopoppe.github.io/function-plot/. Accessed: 2017-06-08.

[7] Google Maps JavaScript API. https://developers.google.com/maps/documentation/javascript/. Accessed: 2017-06-08.

[8] HTML5 - Web developer guides — MDN. https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. Accessed: 2017-09-27.

[9] JavaScript. https://www.javascript.com. Accessed: 2017-06-08.

[10] jQuery. https://jquery.com/. Accessed: 2017-08-13.

[11] MongoDB Overview. https://www.tutorialspoint.com/mongodb/mongodb_overview.htm. Accessed: 2017-10-02.

[12] Morris.js. http://morrisjs.github.io/morris.js. Accessed: 2017-06-08.

[13] MQTT Essentials: Client, Broker and Connection Establishment. https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment. Accessed: 2017-09-16.

[14] MQTT.js. https://www.npmjs.com/package/mqtt. Accessed: 2017-06-08.

[15] Node.js. https://nodejs.org/en. Accessed: 2017-06-08.

[16] Node.js - reasons to use, pros and cons, best practices! — Void Canvas. http://voidcanvas.com/describing-node-js/. Accessed: 2017-09-28.

[17] Raphaël—JavaScript Library. http://dmitrybaranovskiy.github.io/raphael/. Accessed: 2017-09-08.

[18] Rickshaw.js. http://code.shutterstock.com/rickshaw. Accessed: 2017-06-08.

[19] Socket.io. https://socket.io. Accessed: 2017-06-08.

[20] Sunseed EU — Sustainable and robust networking for smart electricity distribution. https://sunseed-fp7.eu/. Accessed: 2017-09-27.

[21] SVG Tutorial. https://www.w3schools.com/graphics/svg_intro.asp. Accessed: 2017-10-18.

[22] What Is MongoDB? — MongoDB. https://www.mongodb.com/what-is-mongodb. Accessed: 2017-10-02.

[23] WindowOrWorkerGlobalScope.setInterval() - Web APIs — MDN. https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval. Accessed: 2017-09-27.

[24] Ethan Brown. *Web Development with Node and Express: Leveraging the JavaScript Stack*. "O'Reilly Media, Inc.", 2014.

[25] Mike Cantelon, Marc Harter, TJ Holowaychuk, and Nathan Rajlich. *Node. js in Action*. Manning, 2014.

[26] Jia Hu, Victor CM Leung, Kun Yang, Yan Zhang, Jianliang Gao, and Shusen Yang. *SMART GRID INSPIRED FUTURE TECHNOLOGIES.* Springer, 2016.

[27] Urban Kuhar. *State estimation in power distribution systems*. PhD thesis, 2017. unpublished thesis.

[28] David Sawyer McFarland. *JavaScript & jQuery: the missing manual*. "O'Reilly Media, Inc.", 2011.

[29] David Sawyer McFarland. *CSS3: the missing manual*. "O'Reilly Media, Inc.", 2012.

[30] Fabio Nelli. *Beginning JavaScript Charts: With JqPlot, D3, and Highcharts*. Apress, 2014.

[31] Jimmy J. Nielsen, Herve Ganem, Ljupco Jorguseski, Kemal Alic, Miha Smolnikar, Ziming Zhu, Nuno K. Pratas, Michal Golinski, Haibin Zhang, Urban Kuhar, Zhong Fan, and Ales Svigelj. Secure real-time monitoring and management of smart distribution grid using shared cellular networks. *Wireless Commun.*, 24(2):1–6, April 2017.

[32] K Purchala, R Belmans, L Exarchakos, and AD Hawkes. Distributed generation and the grid integration issues. *Imperial College London, UK, EUSUSTEL, Work Package*, 3:1–6, 2006.

[33] Philip Russom, David Stodder, and Fern Halper. Real-time data, BI, and analytics. Accelerating Business to Leverage Customer Relations, Competitiveness, and Insights. *TDWI best practices report, fourth quarter*, pages 5–25, 2014.

[34] Čedomir Stefanović, Petar Popovski, Ljupčo Jorgušeski, and Radovan Sernec. Sunseed—an evolutionary path to smart grid comms over converged telco and

energy provider networks. In *Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE), 2014 4th International Conference on*, pages 1–5. IEEE, 2014.